JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 6 (2018) 42-46
Wysłane: 2017-10-12
Przyjęte: 2017-10-19

Porównanie konfiguracji i możliwości bibliotek ORM dla systemu Android

Tomasz Serwin*
Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. W artykule omówiono temat konfiguracji i możliwości bibliotek mapowania relacyjno-obiektowego dla systemu Android. Przedstawiono dotychczasowe porównania ORM dla systemów mobilnych. Opisane zostały sposoby konfiguracji oraz wspierane typy atrybutów obiektów.

Słowa kluczowe: orm; android; sqlite

*Autor do korespondencji.

Adres e-mail: tomaszserwin1993@gmail.com

Comparison of the configuration and capabilities of ORM libraries for Android

Tomasz Serwin*
Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract. The article discusses configuration and capabilities of object-relational mapping libraries for the Android operating system. It also presents previous comparisons of ORMs for mobile systems. Configuration methods and supported mappings from java to database types are described.

Keywords: orm; android; sqlite

*Corresponding author.

E-mail address: tomaszserwin1993@gmail.com

1. Wstęp

System Android, którego pierwsza wersja została opublikowana 21 października 2008 r., jest aktualnie najpopularniejszym systemem operacyjnym na urządzenia mobilne[1]. Jego popularność powoduje powstawanie wielu bibliotek o różnym zastosowaniu.

Jednym z ważnych obszarów jest dostęp i komunikacja z bazą danych. Android do przechowywania danych oferuje bazy SQLite. Dodatkowo zapewniany jest podstawowy zestaw funkcji, który pozwala na komunikację z oferowaną bazą [2].

Jednak nie wszystkich programistów zadowalają możliwości dostarczanego API. Powstały więc liczne biblioteki mapowania relacyjno-obiektowego oferujące rozszerzone możliwości działania na bazie danych.

Badanie przydatności ORM i porównanie ORM pod względem wydajnościowym nie jest nowym tematem, chociaż nie jest on dokładnie zgłębiony w odniesieniu do wszystkich platform. Jednak oprócz wydajności, ważnym elementem jest również wygoda używania biblioteki i jej możliwości.

2. Analiza literatury

Zbigniew Rosiek w artykule „Mapowanie obiektowo relacyjne ORM – czy tylko dobra idea?" przedstawia wady i zalety ORM w oparciu o JPA (Java Persistence API). Wśród problemów, które można napotkać przy korzystaniu z ORM wskazuje on na inne systemy typów używany przez język programowania i przez bazę danych [3]. Ważna jest więc obecność automatycznego mapowania pomiędzy najczęściej spotykanymi typami.

Szereg artykułów omawia porównanie wydajnościowe. Artykuł „A Comparative Study of the features and Performance of ORM Tools in a .NET Environment" obejmuje porównanie narzędzi ORM na platformę .NET. Porównywane są Entity Framework oraz NHibernate. Kryterium porównania jest czas wykonywania różnych operacji Select, Insert oraz Delete [4]. Dodatkowo wskazane zostały również wady i zalety nauki narzędzi ORM.

W artykule „Mobile Data Store Platforms: test Case based Performance Evaluation" autorstwa K. Kussainova i B. Kumalakova dokonano porównania 3 technologii mobilnych: Realm, SnappyDB oraz API Androida. Porównano jest pod względem operacji zapisywania – tworzono 10000 rekordów – jak i odczytywania – tworzono zapytania odczytujące 12000 rekordów [5].

Ciekawe spojrzenie na temat zawiera artykuł „Understanding the Energy, Performance, and Programming Effort Trade-offs of Android Persistence Frameworks". W artykule tym również zbadano wydajność operacji Insert oraz Select, poprzez zmierzenie czasu wykonywania. Dodatkowo jednak zbadano również operacje Update i Delete, oraz dodano badanie wpływu tych operacji przy zastosowaniu różnych ORM na konsumpcję energii urządzenia [6].

Artykuł „Comparing the Performance of Object Databases and ORM Tools" chociaż nie dotyczy bezpośrednio technologii na platformę Android, zawiera porównanie bibliotek ORM dla języka Java. Przedmiotem porównania są czas tworzenia obiektów oraz czas wykonywania zapytania wydobywającego odpowiednie rekordy. To drugie porównanie wykorzystuje szereg różnych zapytań: wykorzystujących operację złączenia tabel, relacje pomiędzy tabelami oraz porównania. W artykule sprawdzono również wpływ tzw. gorących i zimnych startów aplikacji na czas wykonywania poszczególnych zadań [7].

„The Usage and performance of Object Databases compared with ORM tools in a Java environment" autorstwa Mikalea Kopteff porównuje technologie Versant I Hibernate. Technologie porównano te pod względem szybkości wykonywania zapytań jak i sposobu połączenia z bazą oraz tworzenia transakcji. Pokazano również jak wygląda mapowanie klasy przy użyciu plików XML [8].

"Performance Evaluation of Java Based Object Relational Mapping Tools" to kolejny artykuł porównujący biblioteki mapowania relacyjno obiektowego dla języka Java. Porównane zostały takie technologie jak Hibernate, TopLink oraz Ebean. Oprócz porównania czasu wykonywania różnych zapytań, jak w wyżej wymienionych artykułach, praca ta analizuje również języki zapytań jakie dostarcza każda z badanych technologii [9].

W artykule „Energy Efficiency of ORM Approaches: an Empirical Evaluation" zbadano czas wykonywania podstawowych operacji bazodanowych przez technologie Propel i TinyQueries, takich jak odczyt, zapis, aktualizacja i usuwanie rekordów. Drugim kryterium była dodatkowo konsumpcja energii jaką potrzebuje urządzenie, aby wykonać poszczególne operacje [10].

3. Badane technologie

Na potrzeby artykułu zbadano 6 różnych bibliotek, które podzielić można na 3 kategorie:

1) Biblioteki mapowania obiektowo – relacyjnego używające SQLite
   - GreenDAO
   - OrmLite
   - DBFlow
   - SugarORM
2) Biblioteka mapowania obiektowo – relacyjnego oparta o własny typ plików
   - Realm
3) Rozszerzona warstwa bazodanowa
   - SquiDB

Wprawdzie Realm nie jest oparty o SQLite, jednak jego wsparcie zarówno dla Androida, jak i iOS [11] pozwala na wygodniejsze tworzenie wieloplatformowej aplikacji.

Natomiast SquiDB jest przedstawiana przez twórców nie jako pełnoprawny ORM, ale jako warstwa rozszerzająca możliwości SQLite. SquiDB ma na celu przede wszystkim zachowanie elastyczności związanej z bezpośrednim używaniem SQL[12].

4. Środowisko testowe

Do testów wykorzystano telefon Xiaomi Redmi Note z systemem operacyjnym Android 7.1.1

4.1. Badane technologie

Technologie zostały użyte w następujących wersjach:

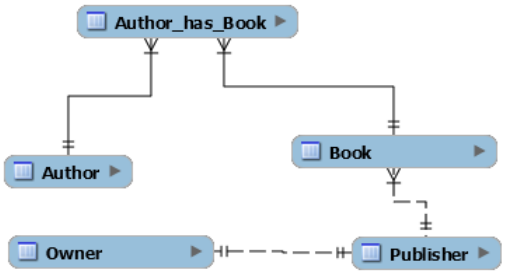
- GreenDAO 3.2.0
- OrmLite 5.0
- DBFlow 4.0.0-beta5
- SugarORM 1.5
- Realm 3.0.0.
- SquiDB 4.0.0-beta.1

4.2. Projekt testowy

Aby porównać technologie, dla każdej z nich utworzono oddzielny projekt. Każdy z projektów zawierał 3 klasy modelu:

- Book, zawierającą informacje o książce
- Author, zawierającą informacje o autorze książki
- Publisher, zawierającą informacje o wydawcy książki
- Owner, zawierającą informacje o właścicielu wydawnictwa.

Klasy połączone zostały relacjami. *Book* i *Publisher* połączone zostały relacją jeden do wielu (jeden wydawca może wydać wiele książek). *Author i Book* połączone zostały relacją wiele do wielu (książka może być napisana przez wielu autorów, autor może napisać wiele książek). Klasy *Owner* i *Publisher* łączy relacja jeden do jednego. Schemat takiej bazy danych zawarty jest na rysunku 1.



Rys 1. Relacje pomiędzy encjami.

Klasy zawierają atrybuty o różnych typach. Celem jest zbadanie, czy są one automatycznie mapowane na typy bazodanowe, czy konwersja musi zostać wykonana ręcznie. Badane typy to:

- typy proste (int, boolean itd.)
- typy boxed (Integer, Boolean itd.)
- String
- Date
- List.

Ponadto utworzono typ wyliczeniowy *Genre,* opisujący gatunek książki, i sprawdzono czy jest on automatycznie mapowany na typ bazodanowy. Jeśli nie, wskazano sposób jak takie mapowanie wykonać.

5. Analiza wyników

W technologiach ORM na klasy można mapować tabele najczęściej na 3 sposoby:

- poprzez adnotacje,
- poprzez dziedziczenie dostarczanej klasy,
- poprzez pliki XML

Pierwsze badanie objęło sprawdzenie, jakie sposoby mapowania umożliwiały badane technologie. Wyniki zamieszczono w tabeli 1.

Tabela 1. Sposób mapowania klas na tabele bazy danych

	Adnotacja	**Dziedziczenie**	**XML**
GreenDAO	Wymagana	Nie	Nie
OrmLite	Wymagana	Nie	Nie
DBFlow	Wymagana	Możliwe	Nie
SugarORM	Nie	Wymagane	Nie
Realm	Możliwe	Możliwe	Nie
SquiDB	Wymagana	Wymagane	Nie

Spośród wszystkich technologii jedynie Realm zapewniało elastyczność wyboru pomiędzy adnotacją, a dziedziczeniem. Mapowanie poprzez adnotację wymagało również implementacji interfejsu *RealmModel*.

DBFlow wymaga adnotowania mapowanych klas, ale zapewnia również możliwość dziedziczenia po klasie *BaseModel*. Pozwala to na wygodne wykonywanie poleceń (save, delete i innych) poprzez wywołanie ich wprost na obiekcie. W przypadku, gdy klasa *BaseModel* nie jest dziedziczona, wywołanie takich metod następuje poprzez adapter.

OrmLite i GreenDAO umożliwiają jedynie adnotowanie mapowanej klasy. Obsługa poleceń wykonywana jest poprzez obiekt DAO, który wykonuje wszystkie operacje związane z rekordami.

SugarORM wymaga dziedziczenia klasy *SugarRecord*. Wymagane dziedziczenie ogranicza elastyczność aplikacji, niemożliwe jest dziedziczenie klasy modelu po innej klasie nadrzędnej.

Ciekawe rozwiązanie zastosowane jest w SquiDB. Do mapowania wykorzystywana jest oddzielna klasa, adnotowana poprzez *@TableModelSpec*. Następnie generator kodu generuje klasę reprezentującą wiersz w tabeli lub wiersz widoku. Klasa reprezentująca wiersz tabeli dziedziczy po *TableModel*, a klasa reprezentująca wiersz widoku po *ViewModel*. Obie klasy dziedziczą po klasie *AbstractModel*. Rozwiązanie to pozwala na elastyczność w przypadku klasy adnotowanej (może ona dziedziczyć po innej klasie) i jednocześnie ułatwia wykonywanie operacji poprzez generowane klasy.

Co ciekawe, żadna z technologii nie umożliwia mapowania poprzez pliki XML. Jest to zaskoczenie, gdyż najpopularniejszy ORM dla języka Java, Hibernate [13], zapewnia taką możliwość. Pozwala to na całkowite oddzielenie kodu mapującego od klasy modelu, przez co nie jest ona zanieczyszczona niepotrzebnym kodem.

Drugie badanie miało na celu sprawdzenie dostępności adnotacji konfigurujących. Wyniki zostały zawarte w tabeli 2.

Tabela 2. Dostępność adnotacji konfigurujących

	Tabela	**Id**	**Kolumny**	**Jeden do jednego**	**Jeden do wielu**	**Wiele do wielu**
GreenDAO	Tak	Tak	Tak	Tak	Tak	Tak
OrmLite	Tak	Tak	Tak	Tak	Tak	Nie
DBFlow	Tak	Tak	Tak	Tak	Tak	Tak
SugarORM	Nie	Nie	Nie	Nie	Nie	Nie
Realm	Tak	Nie	Nie	Nie	Nie	Nie
SquiDB	Tak	Tak	Tak	Nie	Nie	Nie

Brak danej adnotacji nie przesądza jednak o braku funkcjonalności. Używając ORMLite i DBFlow należy zaznaczyć adnotacją, które z atrybutów mają być mapowane, pozostałe są zaś ignorowane. Odwrotna sytuacja jest w przypadku Realm, SugarORM, SquiDB i GreenDAO: w tych technologiach adnotacją należy oznaczyć te pola, które mają zostać zignorowane. Adnotacje oznaczające kolumny w GreenDAO i SquiDB pozwalają na dodatkowe określenie parametrów np. nazwa pola w bazie danych.

Realm wprawdzie nie posiada adnotacji do oznaczania relacji, jednak są one możliwe do wykonania. Wystarczy ustawić obiekt lub listę obiektów (*RealmList*) jako parametr klasy, a odwzorowane zostanie połączenie jeden do jednego, jeden do wielu lub wiele do wielu. Podobnie zachowuje się SugarORM. SquiDB nie posiada adnotacji, które ułatwiłyby modelowanie relacji, nie tworzy też tych relacji automatycznie. Proponowanym przez autorów rozwiązaniem [12] jest przechowywanie identyfikatora rekordu, z którym aktualny rekord jest związany i użycie *ModelMethod* lub funkcji pomocniczych. Mają one wywołać odpowiednie zapytanie na bazie danych.

GreenDAO, OrmLite i DBFlow posiadają adnotacje wspomagające tworzenie relacji wiele do wielu. Wszystkie osiągają ten efekt poprzez klasę mapowaną na tabelę pośredniczącą. DBFlow sam automatycznie generuje taką klasę, pozostałe technologie wymagają od programisty utworzenia takich klas samodzielnie.

Spośród wybranych technologii tylko SugarORM nie pozwala na oznaczenie encji poprzez adnotację i wymaga dziedziczenia po klasie *SugarRecord<T>*.

Ostatnie badanie dotyczyło mapowania typów z języka Java na języki bazodanowe. Wyniki badania przedstawiono w tabeli 3.

Wsparcie dla typów prostych, typów boxed, String i Date jest podstawowym wymaganiem w stosunku do ORM i wszystkie wybrane technologie obsługują takie mapowania.

Tabela 3. Typy Java mapowane na typy bazodanowe

	Typy proste	Typy boxed	String	Date	List	Enum
GreenDAO	Tak	Tak	Tak	Tak	Nie	Nie
OrmLite	Tak	Tak	Tak	Tak	Nie	Tak
DBFlow	Tak	Tak	Tak	Tak	Nie	Tak
SugarORM	Tak	Tak	Tak	Tak	Nie	Tak
Realm	Tak	Tak	Tak	Tak	Nie	Nie
SquiDB	Tak	Tak	Tak	Tak	Nie	Tak

Problem następuje w przypadku próby mapowania prostej listy, gdyż takiego mapowania nie wykonuje żadne z narzędzi – mapowane są tylko listy innych encji. Rozwiązaniem tego problemu może być przekształcanie listy do postaci JSON a następnie umieszczanie takiego ciągu znaków w bazie danych jako varchar. Nie pozwala to jednak na wygodne przeszukiwanie tak zapisanych informacji w bazie danych.

Nie wszystkie technologie automatycznie mapują typ wyliczeniowy (enum) do bazy danych. Najlepiej wywiązuje się z tego OrmLite, który pozwala zarówno na mapowanie enum jako tekstu (varchar) jak i mapowania ordinal [14] – unikatowego numeru danego elementu typu wyliczeniowego. Automatyczne mapowanie wykonują również SugarORM, SquiDB oraz DBFlow.

Dokumentacje pozostałych technologii wskazują jednak na rozwiązanie takiego problemu. GreenDAO proponuje utworzenie konwertera [15] (przykład 1), a w Realm rozwiązaniem jest zapisywanie typu wyliczeniowego jako String (przykład 2).

Przykład 1. Mapowanie typu wyliczeniowego w GreenDAO

```
@Entity
public class User {
@Id
private Long id;
@Convert(converter = RoleConverter.class, columnType =
String.class)
private Role role;
enum Role {
DEFAULT, AUTHOR, ADMIN
}
static class RoleConverter implements
PropertyConverter<Role, String> {
@Override
public Role convertToEntityProperty(String databaseValue) {
return Role.valueOf(databaseValue);
}
@Override
public String convertToDatabaseValue(Role entityProperty) {
return entityProperty.name();
}}}
```

Realm proponuje rozwiązanie podobne do tego, jakie OrmLite wykonuje automatycznie, czyli konwertowanie enum do String i przechowywanie jako varchar.

Przykład 2. Mapowanie typu wyliczeniowego w Realm

```
public enum MyEnum {
FOO, BAR;
}
public class Foo extends RealmObject {
private String enumDescription;
public void saveEnum(MyEnum val) {
this.enumDescription = val.toString();
}
public MyEnum getEnum() {
return MyEnum.valueOf(enumDescription);}}
```

6. Wnioski

Przy wyborze odpowiedniego ORM warto sprawdzić, czy udostępnia on odpowiednie możliwości.

Programiści używający Hibernate w projektach Java docenią GreenDAO i OrmLite. Technologie te bardzo przypominają Hibernate. Udostępniają konfigurację poprzez adnotację, wspierają wszystkie podstawowe typy (chociaż w przypadku GreenDAO mapowanie enum trzeba wykonać samodzielnie). Umożliwiają również łatwe mapowanie relacji. GreenDAO zapewnia taka możliwość dla wszystkich typów relacji – 1:1, 1:n, n:m – a OrmLite pozwala na osiągnięcie efektu relacji n:m poprzez samodzielne stworzenie tabeli pośredniczącej.

Atrakcyjnym wyborem jest również DBFlow, które oprócz możliwości dostarczanych przez dwie wcześniej wspomniane technologie zapewnia również możliwość dziedziczenia encji po specjalnej klasie. Ta specjalna klasa powoduje, że ułatwione jest wykonywanie operacji CRUD (Create Read Update Delete), gdyż wywoływane są one z obiektu – niepotrzebna jest specjalna klasa DAO.

Alternatywą dla wymienionych bibliotek może być Realm. Nie zapewnia on wprawdzie tak bogatej gamy adnotacji, jednak w tym przypadku są one zbędne. Realm sam wykrywa atrybuty do mapowania (tylko te niepotrzebne należy ignorować) i łączy tabele relacjami, jeśli atrybutem encji jest inna encja lub lista encji. Zaletą jest również wieloplatformowość – z tego ORM można skorzystać zarówno na systemie Android jak i iOS.

SugarORM jest zdecydowanie najmniej atrakcyjna opcją. Wymaga dziedziczenia, brakuje mu mechanizmu mapowania enum, jego lista adnotacji jest bardzo uboga. Jest to ORM, który może mieć zastosowanie najwyżej przy prostych projektach, gdyż brakuje mu zalet pozostałych produktów.

W oddzielnej kategorii można sklasyfikować SquiDB. Jest to technologia o szerokich możliwościach, ale wymaga większych nakładów pracy. Pomimo, że nie zapewnia mechanizmu mapowania relacji, to umożliwia wykonanie takiego mapowania samodzielnie, dzięki czemu programista ma kontrolę nad mechanizmem od jego początku, do końca.

Literatura

[1] http://www.gartner.com/newsroom/id/3609817 Gartner Says Worldwide Sales of Smartphones Grew 7 Percent in the Fourth Quarter of 2016 [15.02.2017]

[2] https://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html SQLiteDatabase [20.05.2017]

[3] Z. Rosiek, Mapowanie obiektowo relacyjne (ORM) – czy tylko dobra idea?, Zeszyty Naukowe Warszawskiej Wyższej Szkoły Informatyki, nr 4, 2010.

[4] S. Cvetković, D. Janković, A Comparative Study of the Features and Performance of ORM Tools in a .NET Environment, W: Dearle A., Zicari R.V. (eds) Objects and Databases. ICOODB 2010. Lecture Notes in Computer Science, vol 6348. Springer, Berlin, Heidelberg.

[5] K. Kussainov, B. Kumalakov, Mobile Data Store Platforms: Test Case based Performance Evaluation, 8th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management, At Porto, Portugal, Volume: 3.

[6] J. Pu, Z. Song, E. Tilevich, Understanding the Energy, Performance, and Programming Effort Trade-Offs of Android Persistence Frameworks, 2016 IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS).

[7] P. Van Zyl, D.G. Kourie, A. Boake, Comparing the Performance of Object Databases and ORM Tools, Proceeding of South African Institute for Computer Scientists and Information Technologists, 1-11, 2006.

[8] . M. Kopteff, "The usage and performance of object databases compared with orm tools in a java environment." [Online]. http://www.odbms.org/wp-content/uploads/2013/11/045.01-Kopteff-TheUsage-and-Performance-of-Object-Databases-Compared-with-ORM-Tools-ina-Java-Environment-March-2008.pdf.

[9] S.N. Bhatti, Z.H. Abro, F. Rufabro, Performance Evaluation of Java Based Object Relational Mapping Tools, Mehran University Research Journal of Engineering and Technology, 32(2):159--166, 2013.

[10] G. Procaccianti, P. Lago, W. Diesveld, Energy Efficiency of ORM Approaches: an Empirical Evaluation, Proceeding ESEM '16 Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement.

[11] https://realm.io/products/realm-mobile-database/ Realm Mobile Database [20.05.2017]

[12] https://github.com/yahoo/squidb Oficjalna strona projektu SquiDB [10.07.2017]

[13] https://blogs.oracle.com/theaquarium/dzone-survey-shows-jpa-dominates-java-persistence DZone Survey Shows JPA Dominates Java Persistence [23.10.2015]

[14] http://ormlite.com/javadoc/ormlite-core/doc-files/ormlite_2.html#Persisted-Types Dokumentacja OrmLite, Persisted Data Types [15.09.2017]

[15] http://greenrobot.org/greendao/documentation/custom-types/ Dokumentacja GreenDAO, Custom Types [15.09.2017].